```
Help on module dspmodule:

NAME
    dspmodule

DESCRIPTION
    DSP Library for solving basic Digital Signal Processing Experiments;
    Paper Code - EC692
    Developed by Mr. Sujoy Mondal, Asst. Professor, Dept of ECE, RCCIIT

FUNCTIONS
    autocorr(var1)
        Autocorrelation of a sequence
        --------------------------------
        Input:
        var1 = signal 1 [1-D numpy array],
        --------------------------------
        Output:
        out = output signal [1-D numpy array]

    butterorder(wp, wst, Ap, As)
        Order of analog prototype filter
        --------------------------------
        Input:
        wp = passband freq (rad/sec) = scalar - LPF & HPF; 1-D numpy array - BSF & BPF
        wst = stopband freq (rad/sec) = scalar - LPF & HPF; 1-D numpy array - BSF & BPF,
        Ap = passband attenuation (max) in dB,
        As = stopband attenuation (min) in dB
        --------------------------------
        Output:
        N - order of analog filter,
        wc - cutoff freq (rad/sec) = scalar for LPF & HPF; 1-D numpy array for BSF & BPF

    circonv(sig1, sig2, plotflag=True)
        Circular convolution
        --------------------------------
        Input:
        sig1 = signal 1 [1-D numpy array],
        sig2 = signal 2 [1-D numpy array],
        plotflag=[True]
        --------------------------------
        Output:
        y = output [1-D numpy array]

    coeff2freq_response(b, a, N=512)
        Coefficients to Frequency Response
        --------------------------------
        Input:
        b = num [1-D numpy array],
        a = den [1-D numpy array],
        N=[512] = sample points
        --------------------------------
        Output:
        w = digital angular freq (rad/samples) [0,pi]
        H = freq response [1-D numpy array]

    crosscorr(var1, var2)
        Crosscorrelation of two sequences
        --------------------------------
        Input:
        var1 = signal 1 [1-D numpy array],
        var2 = signal 2 [1-D numpy array],
        --------------------------------
        Output:
        out = output signal [1-D numpy array]

    dft(sig, N=None, win=0)
        DFT computation
        --------------------------------
```

```
    Input:
    sig = signal [1-D numpy array],
    N=[None] = DFT point
    win=[0] = window type = 0/1/2/3 [rec, bartlett, hamm, hann]
    ---------------------------------
    output:
    X dft = spectrum [1-D numpy array]

firdesign(tap, Fs, Fc, win='hann', fir_type='lowpass')
    FIR Filter Design
    ---------------------------------
    Input:
    tap = Tapping points
    Fs = Sampling Freq (Hz)
    Fc = cutoff_freq (Hz) = [scalar] - LPF & HPF [1-D numpy array] - BPF & BSF,
    win=['hann'] = 'hann/blackman/hamming/bartlett/boxcar[rectangular]',
    fir_type='lowpass/highpass/bandpass/bandstop'
    ---------------------------------
    Output:
    b = num coeff [1-D numpy array]
    a = den coeff [1-D numpy array]

idft(spec)
    IDFT computation
    ---------------------------------
    Input:
    spec = spectrum [1-D numpy array]
    ---------------------------------
    Output:
    x = signal [1-D numpy array]

iirbutterdesign(N, wc, Fs, iir_type='lowpass')
    Butterworth IIR design
    ---------------------------------
    Input:
    N = order of analog filter,
    wc = cutoff freq (rad/sec) = scalar for LPF & HPF; 1-D numpy array for BSF & BPF ,
    Fs = sampling frequency (Hz),
    iir_type=['lowpass'] = type of filter = 'lowpass/highpass/bandpass/bandstop'
    ---------------------------------
    Output:
    b = num coeff [1-D numpy array],
    a = den coeff [1-D numpy array]

linearconv(sig1, sig2, Ix=0, Ih=0, mode=1, plotflag=True)
    Linear convolution between two sequences
    ---------------------------------
    Input:
    sig1 = signal 1 [1-D numpy array],
    sig2 = signal 2 [1-D numpy array],
    Ix=[0] = start idx of signal 1,
    Ih=[0] = start idx of signal 2,
    mode=[1] = mode selection = 1/2 [full, same=max(Lx, Lh)],
    plotflag=[True]
    ---------------------------------
    Output:
    y = output [1-D numpy array]

linearfilter(b, a, x)
    Linear Filtering
    ---------------------------------
    Input:
    b = num [1-D numpy array]
    a = den [1-D numpy array]
    x = input signal [1-D numpy array]
    ---------------------------------
    Output:
    y = output signal [1-D numpy array]
```

```
polezero2freq_response(Z, P, K=1, N=512)
    Pole Zero to Frequency Response
    --------------------------------
    Input:
    Z = zeros [1-D numpy array],
    P = poles [1-D numpy array],
    K=[1] = gain
    N=[512] = sample points
    --------------------------------
    Output:
    w = digital angular freq (rad/samples) [1-D numpy array] [0,pi]
    H = freq response [1-D numpy array]

tf2polezero(b, a)
    Coefficients to Pole Zero
    --------------------------------
    Input:
    b = num [1-D numpy array]
    a  den [1-D numpy array]
    --------------------------------
    Output:
    Z = zeros [1-D numpy array],
    P = poles [1-D numpy array],
    K = gain
```